wirepy Documentation

Release

Lukas Lueg

January 26, 2014

Contents

Wirepy aims to remedy the disastrous situation of packet-dissection-libraries available to the Python programming language. It is a foreign function interface to use Wireshark within Python as implemented by CPython and PyPy.

The currently available options are either painfully slow or lack features. Wireshark provides support for more than 1.300 protocols, more than 125.000 fields within those protocols and more than 1.500.000 defined values and is actively maintained.

Get the code from GitHub.

Note: This library is created out of pure necessity. I dont' know know where it is headed or even feasible to create a direct binding to `libwireshark`. The best current source of documentation are the unittests.

CHAPTER 1

Module reference

1.1 `cdata` Module

Helper module to make working with CFFI more convenient.

Classes that mainly wrap c-like *struct* may subclass `CDataObject` which carries `MetaCDataObject` as it's metaclass. When a deriving class is created, all class-level attributes that derive from `BaseAttribute` are replaced with standard python properties that access the wrapped struct-members, automatically cast to python types, raise Exceptions and keep references to allocated memory in order to handle garbage collection.

Note: It's not clear wether to keep this module at all, the overhead during runtime is probably significant. It does however provide convenience until design decisions quite down.

exception `wirepy.lib.cdata.`**`AttributeAccessError`**

Bases: `builtins.AttributeError`

Indicates access to an attribute that can't be accessed that way.

`__weakref__`

list of weak references to the object (if defined)

exception `wirepy.lib.cdata.`**`AttributeSizeError`**

Bases: `builtins.AttributeError`

A list-like attribute was set to an incorrect size.

`__weakref__`

list of weak references to the object (if defined)

class `wirepy.lib.cdata.`**`Attribute`**(*structmember=None*, *can_read=None*, *can_write=None*, *can_del=None*, *doc=None*)

Bases: `wirepy.lib.cdata.BaseAttribute`

An basic attribute that sets and gets the raw value.

class `wirepy.lib.cdata.`**`BaseAttribute`**(*structmember=None*, *can_read=None*, *can_write=None*, *can_del=None*, *doc=None*)

Bases: `builtins.object`

An attribute on a cdata-object.

An attribute defines methods to read, write and delete values. These methods end up as property()s on the final class.

`__init__`(*structmember=None*, *can_read=None*, *can_write=None*, *can_del=None*, *doc=None*)

Parameters

- **structmember** – Name of the member to access by this attribute. `MetaCDataObject` will use the attribute's name in case **structmember** is None.
- **can_read** – Indicates wether this attribute should provide read access to the underlying member or raise an `AttributeAccessError`.
- **can_write** – Same as **can_read** for write access.
- **can_del** – Sam as **can_del** for deletion.
- **doc** – docstring to be placed on the final property.

deleter()

Generate a function that serves as a deleter.

deleter_cant_delete()

Generate a function that indicates an access-error while deleting

getter()

Generate a function that serves as a getter.

getter_cant_get()

Generate a function that indicates an access-error while reading.

setter()

Generate a function that serves as a setter.

setter_cant_set()

Generate a function that indicates an access-error while writing.

__weakref__

list of weak references to the object (if defined)

class `wirepy.lib.cdata.`**BooleanAttribute**(*structmember=None*, *can_read=None*, *can_write=None*, *can_del=None*, *doc=None*)

Bases: `wirepy.lib.cdata.BaseAttribute`

A boolean value.

class `wirepy.lib.cdata.`**CDataObject**

Bases: `builtins.object`

Base class for objects wrapping *struct*

__weakref__

list of weak references to the object (if defined)

class `wirepy.lib.cdata.`**IntListAttribute**(*sizeattr*, **args*, ***kwargs*)

Bases: `wirepy.lib.cdata.ListAttribute`

A list of integers like "int*".

A new int[] is created and kept upon assigning to the attribute.

class `wirepy.lib.cdata.`**ListAttribute**(*sizeattr*, **args*, ***kwargs*)

Bases: `wirepy.lib.cdata.BaseAttribute`

A list-like attribute, such as "char **" or "int*"

class `wirepy.lib.cdata.`**MetaCDataObject**

Bases: `builtins.type`

Metaclass that automatically creates accessors to the underlying c-level *struct*.

A class using this metaclass should define a single "_struct" attribute that names the to-be-wrapped *struct*. All instances of objects deriving from `BaseAttribute` are **replaced** by standard python properties that may keep a reference to their `BaseAttribute`- instance. Instances of such class should have a instance-attribute named "cdata" that references an instance of the wrapped *struct*.

class `wirepy.lib.cdata.`**`ROAttribute`**(*structmember=None*, *can_read=None*, *can_write=None*, *can_del=None*, *doc=None*)
Bases: `wirepy.lib.cdata.Attribute`

A basic attribute that can only read but never write.

class `wirepy.lib.cdata.`**`ROStringAttribute`**(*structmember=None*, *can_read=None*, *can_write=None*, *can_del=None*, *doc=None*)
Bases: `wirepy.lib.cdata.StringAttribute`

A zero-terminated string that can only be read but never be written.

class `wirepy.lib.cdata.`**`StringAttribute`**(*structmember=None*, *can_read=None*, *can_write=None*, *can_del=None*, *doc=None*)
Bases: `wirepy.lib.cdata.BaseAttribute`

A null-terminated string.

1.2 `column` Module

Wireshark displays generic information about a packet's content in it's GUI using a set of columns. Each column has one of several pre-defined column-types which `libwireshark` knows about and fills with content while dissecting a packets. This allows dissectors of all kinds to provide information about a packet, no matter where in the protocol this information is ultimately retrieved from.

For example, `Type.PROTOCOL` provides the name of the deepest protocol found within a frame; a raw ethernet frame may provide "eth" for PROTOCOL, a IP packet within the ethernet packet overrules this to "ip", a TCP packet within the IP-packet again overrules to 'tcp' and a HTTP packet within the TCP packet finally overrules to 'http'.

Note: Wireshark uses columns in concert with it's preferences, the API reading column-settings directly from the global preferences object. To make this concept more flexible, we avoid this binding.

exception `wirepy.lib.column.`**`ColumnError`**
Bases: `builtins.Exception`

Base class for all column-related errors.

`__weakref__`
list of weak references to the object (if defined)

exception `wirepy.lib.column.`**`InvalidColumnType`**
Bases: `wirepy.lib.column.ColumnError`

An invalid column-type was provided.

class `wirepy.lib.column.`**`Format`**(*type_=None*, *init=None*, *title=None*, *custom_field=None*, *custom_occurrence=None*, *visible=None*, *resolved=None*)
Bases: `wirepy.lib.cdata.CDataObject`

A fmt_data

`__init__`(*type_=None*, *init=None*, *title=None*, *custom_field=None*, *custom_occurrence=None*, *visible=None*, *resolved=None*)

param init: The underlying fmt_data-object to wrap or None to create a new one.

custom_field
Field-name for custom columns.

custom_occurrence
Optional ordinal of occcurrence of the custom field.

resolved
True to show a more human-readable name.

title
Title of the column.

type_
The column's type, one of `Type`.

visible
True if the column should be hidden in GUI.

class `wirepy.lib.column.`**`Type`** (*fmt*)
Bases: `builtins.object`

A column-type.

ABS_DATE_TIME
Absolute date and time

alias of `COL_ABS_DATE_TIME`

ABS_TIME
Absolute time

alias of `COL_ABS_TIME`

BSSGP_TLLI
!! DEPRECATED !! - GPRS BSSGP IE TLLI

alias of `COL_BSSGP_TLLI`

CIRCUIT_ID
Circuit ID

alias of `COL_CIRCUIT_ID`

COS_VALUE
!! DEPRECATED !! - L2 COS Value

alias of `COL_COS_VALUE`

CUMULATIVE_BYTES
Cumulative number of bytes

alias of `COL_CUMULATIVE_BYTES`

CUSTOM
Custom column (any filter name's contents)

alias of `COL_CUSTOM`

DCE_CTX
DCE/RPC connection orientated call id OR datagram sequence number

alias of `COL_DCE_CTX`

DEF_DL_DST
Data link layer destination address

alias of `COL_DEF_DL_DST`

DEF_DL_SRC
Data link layer source address

alias of `COL_DEF_DL_SRC`

DEF_DST
Destination address

alias of `COL_DEF_DST`

DEF_DST_PORT
Destination port

alias of `COL_DEF_DST_PORT`

DEF_NET_DST
Network layer destination address

alias of `COL_DEF_NET_DST`

DEF_NET_SRC
Network layer source address

alias of `COL_DEF_NET_SRC`

DEF_SRC
Source address

alias of `COL_DEF_SRC`

DEF_SRC_PORT
Source port

alias of `COL_DEF_SRC_PORT`

DELTA_TIME
Delta time

alias of `COL_DELTA_TIME`

DSCP_VALUE
IP DSCP Value

alias of `COL_DSCP_VALUE`

EXPERT
Expert info

alias of `COL_EXPERT`

FREQ_CHAN
IEEE 802.11 (and WiMax?) - Channel

alias of `COL_FREQ_CHAN`

FR_DLCI
!! DEPRECATED !! - Frame Relay DLCI

alias of `COL_FR_DLCI`

HPUX_SUBSYS
!! DEPRECATED !! - HP-UX Nettl Device ID

alias of `COL_HPUX_SUBSYS`

IF_DIR
FW-1 monitor interface/direction

alias of COL_IF_DIR

INFO
Description

alias of COL_INFO

NUMBER
Packet list item number

alias of COL_NUMBER

class NUM_COL_FMTS (**args*, ***kwargs*)
Bases: builtins.Mock

Command line specific time (default relative)

Type.**OXID**
!! DEPRECATED !! - Fibre Channel OXID

alias of COL_OXID

Type.**PACKET_LENGTH**
Packet length in bytes

alias of COL_PACKET_LENGTH

Type.**PROTOCOL**
Protocol

alias of COL_PROTOCOL

Type.**REL_CONV_TIME**
blurp

alias of COL_REL_CONV_TIME

Type.**REL_TIME**
Relative time

alias of COL_REL_TIME

Type.**REST_DST**
Resolved destination

alias of COL_RES_DST

Type.**REST_DST_PORT**
Resolved destination port

alias of COL_RES_DST_PORT

Type.**RES_DL_DST**
Unresolved DL destination

alias of COL_RES_DL_DST

Type.**RES_DL_SRC**
Resolved DL source

alias of COL_RES_DL_SRC

Type.**RES_NET_DST**

Resolved net destination

alias of `COL_RES_NET_DST`

Type.**RES_NET_SRC**

Resolved net source

alias of `COL_RES_NET_SRC`

Type.**RES_SRC**

Resolved source

alias of `COL_RES_SRC`

Type.**RES_SRC_PORT**

Resolved source port

alias of `COL_RES_SRC_PORT`

Type.**RSSI**

IEEE 802.11 - received signal strength

alias of `COL_RSSI`

Type.**RXID**

!! DEPRECATED !! - Fibre Channel RXID

alias of `COL_RXID`

Type.**SRCIDX**

!! DEPRECATED !! - Dst port idx - Cisco MDS-specific

alias of `COL_SRCIDX`

Type.**TEI**

Q.921 TEI

alias of `COL_TEI`

Type.**TX_RATE**

IEEE 802.11 - TX rate in Mbps

alias of `COL_TX_RATE`

Type.**UNRES_DL_DST**

Unresolved DL destination

alias of `COL_UNRES_DL_DST`

Type.**UNRES_DL_SRC**

Unresolved DL source

alias of `COL_UNRES_DL_SRC`

Type.**UNRES_DST**

Unresolved destination

alias of `COL_UNRES_DST`

Type.**UNRES_DST_PORT**

Unresolved destination port

alias of `COL_UNRES_DST_PORT`

Type.**UNRES_NET_DST**
Unresolved net destination

alias of COL_UNRES_NET_DST

Type.**UNRES_NET_SRC**
Unresolved net source

alias of COL_UNRES_NET_SRC

Type.**UNRES_SRC**
Unresolved source

alias of COL_UNRES_SRC

Type.**UNRES_SRC_PORT**
Unresolved source Port

alias of COL_UNRES_SRC_PORT

Type.**UTC_DATE_TIME**
UTC date and time

alias of COL_UTC_DATE_TIME

Type.**UTC_TIME**
UTC time

alias of COL_UTC_TIME

Type.**VSAN**
VSAN - Cisco MDS-specific

alias of COL_VSAN

Type.**__init__**(*fmt*)
Get a reference to specific column-type.

Parameters **fmt** – One of the defined column-types, e.g. Number

classmethod Type.**iter_column_formats**()
Iterate over all available column formats.

Returns An iterator that yields instances of Type.

Type.**__weakref__**
list of weak references to the object (if defined)

1.3 dfilter Module

Wireshark uses display filters for packet filtering within the GUI. The rich syntax makes them very useful for filtering packets without manual inspection of a packet's protocol tree. Because display filters are compiled to bytecode and executed within wireshark's own VM, complex filters also perform much better than inspection from within Python.

See the official documentation for for information about their syntax.

Example:

```
# wt is a wtap.WTAP-instance, frame is a epan.Frame-instance
filter_islocal = dfilter.DisplayFilter('ip.src==192.168.0.0/16')
edt = epan.Dissect()
edt.prime_dfilter(filter_islocal)
edt.run(wt, frame)
```

```
passed = filter_islocal.apply_edt(edt)
if passed:
    ...
```

exception `wirepy.lib.dfilter.`**`DisplayFilterError`**

Bases: `builtins.Exception`

Base-class for display-filter-related errors

`__weakref__`

list of weak references to the object (if defined)

class `wirepy.lib.dfilter.`**`DisplayFilter`**(*init*)

Bases: `builtins.object`

A display-filter

`__init__`(*init*)

Create a new or wrap an existing struct.

Parameters **init** – A dfilter_t-object or a string

Raises `DisplayFilterError` in case a string was supplied and the new display filter failed to compile.

`apply`(*proto_tree*)

Apply this DisplayFilter to a ProtoTree-instance

`apply_edt`(*edt*)

Apply this DisplayFilter to a Dissect-instance

`dump`()

Print bytecode to stdout

`prime_proto_tree`(*proto_tree*)

Prime a ProtoTree-instance using the fields/protocols used in this DisplayFilter

`__weakref__`

list of weak references to the object (if defined)

1.4 `dumpcap` Module

To capture network traffic from live interfaces the external `dumpcap`- program is used (as in `tshark` and `wireshark`). This module provides classes and functions to deal with `dumpcap` and get useful results from it.

exception `wirepy.lib.dumpcap.`**`BadFilterError`**

Bases: `wirepy.lib.dumpcap.ChildError`

`dumpcap` reports that the given capture filter could not be compiled.

exception `wirepy.lib.dumpcap.`**`BrokenPipe`**

Bases: `wirepy.lib.dumpcap.DumpcapError`

The communication-pipe to `dumpcap` was closed or the receiving thread has died because it received an unexpected message from `dumpcap`.

exception `wirepy.lib.dumpcap.`**`ChildError`**

Bases: `wirepy.lib.dumpcap.DumpcapError`

`dumpcap` has reported an error or died with a process exit status indicating failure.

exception `wirepy.lib.dumpcap.`**`DumpcapError`**

Bases: `builtins.Exception`

Base-class for all exceptions

`__weakref__`

list of weak references to the object (if defined)

exception `wirepy.lib.dumpcap.`**`NoEvents`**

Bases: `wirepy.lib.dumpcap.DumpcapError`

No events are available from `dumpcap` while waiting on a blocking call.

class `wirepy.lib.dumpcap.`**`CaptureSession`**(***extra_capture_args*)

Bases: `builtins.object`

Use `dumpcap` to capture network traffic from live interfaces.

A new subprocess is created on instantiation which starts immediately. `dumpcap` writes captured traffic to one or more files and reports it's activity through a set of messages. Incoming messages are received by an internal thread that puts *events* on a FIFO-queue were they can be received by calling `wait_for_event()`. One may register an eventhandler-function through `register_eventhandler()` that automatically reacts to certain event-types when `wait_for_unhandled_event()` is called.

The first event after instantiation should be `SP_FILE`, indicating that `dumpcap` has started writing captured traffic. After that, multiple events of type `SP_PACKET_COUNT` arrive to indicate that a number of new packets have been written to the current file.

For example:

```
def print_packet_count(n):
    """Handle new packets as they are written to the current file."""
    # not entirely obvious example on using nonlocal...
    nonlocal fname, cap
    print('%s: %i new, %i in all files' % (fname, n, cap.packetcount))

with CaptureSession(interfaces=('any', ),
                    autostop_duration=30) as cap:
    cap.register_eventhandler(cap.SP_PACKET_COUNT, print_packet_count)
    try:
        # Wait for the first filename
        event_type, event_msg = cap.wait_for_unhandled_event(timeout=10)
        if event_type != cap.SP_FILE:
            # Pipe is out of sync, just exit in any case
            raise RuntimeError
    except NoEvents:
        # Dumpcap did not start capturing for some reason.
        raise RuntimeError('Giving up on dumpcap')
    fname = event_msg
    # Now loop while dumpcap keeps sending messages
    while True:
        print('Switched to file %s' % (fname, ))
        for event_type, event_msg in cap:
            if event_type == cap.SP_FILE:
                # Switch files
                fname = event_msg
                break
        else:
            # The event-iterator stops when dumpcap closes on its own.
            break
```

`__enter__`()

Returns the instance itself

`__exit__`(*exc_type*, *exc_value*, *traceback*)

Kill `dumpcap` through a call to `terminate()` and block until the message-pipe has stopped.

`__init__`(***extra_capture_args*)

Start a new packet capture using `dumpcap`.

Parameters

- **interfaces** – Tuple of interface-names to capture on.
- **capture_filter** – Packet filter to libpcap filter syntax to use while capturing. See the documentation for more information.
- **snaplen** – Packet snapshot length.
- **promiscuous** – Capture in promiscuous-mode (True by default).
- **monitor_mode** – Capture in monitor-mode if available (False by default).
- **kernel_buffer_size** – Size of kernel buffer in MiB.
- **link_layer_type** – Link layer type.
- **wifi_channel** – Set channel on wifi interface to <freq>,[type] if possible.
- **max_packet_count** – Stop capturing after this number of packets.
- **autostop_duration** – Stop capturing after this number of seconds.
- **autostop_filesize** – Stop capturing after this number of KB.
- **autostop_files** – Stop capturing after this number of files.
- **savefile** – Name of file to save to (defaults to a temporary file).
- **group_access** – Enable group read access on the output file(s). (Defaults to False.)
- **ringbuffer_duration** – Switch to next file after this number of seconds.
- **ringbuffer_filesize** – Switch to next file after this number of KB.
- **ringbuffer_files** – Start replacing after this number of files.
- **use_pcapng** – Use pcapng format instead of pcap (Defaults to True).
- **use_libpcap** – Use libpcap format instead of pcapng (Defaults to False).
- **max_buffered_packets** – Maximum number of packets buffered within `dumpcap`.
- **max_buffered_bytes** – Maximum number of bytes used for buffering packets within `dumpcap`.
- **separate_threads** – Use a separate thread per interface (Defaults to False).

The events `SP_ERROR_MSG` and `SP_BAD_FILTER` have handlers automatically registered on them to raise `ChildError` and `BadFilterError` in `wait_for_unhandled_event()`.

`__iter__`()

Iterate over all events received from `dumpcap` until it exits or dies (in which case an exception is raised). The iterator uses `wait_for_unhandled_event()` and blocks until unhandled events arrive.

`register_eventhandler`(*event_type*, *func*)

Register a function to automatically handle an event.

The given function is called by `wait_for_unhandled_event()` with the event-message being the only parameter. One event-type can only have one handler registered at a time.

Parameters

- **event_type** – One of *SP_...* like `CaptureSession.SP_FILE`
- **func** – A callable that will receive the event-message as it's only argument.

stop()

Signal dumpcap to stop capturing and exit.

terminate()

Kill dumpcap.

wait()

Wait until `dumpcap` has ended on its own.

wait_for_event(*block=True*, *timeout=None*)

Wait for events from `dumpcap`.

Parameters

- **block** – If True, the call blocks until an event appears through the pipe.
- **timeout** – The number of seconds a call should block if **block** is True.

Raises `ChildError` if `dumpcap` has died while waiting for events. `BrokenPipe` in case the thread receiving messages from `dumpcap` has died. `NoEvents` if **block** is False and no event is readily available or **block** is True and the timeout-time has passed.

Returns A tuple of *(event_type, event_msg)*.

wait_for_unhandled_event(*block=True*, *timeout=None*)

Wait for events from `dumpcap` and pass them to their respective event-handler.

Returns the next event that has no handler registered. See `CaptureSession.wait_for_event()` for details on the parameters and the return values.

Any exceptions raised by registered event-handlers are reported to the caller.

SP_BAD_FILTER = 66

The supplied capture filter failed to compile; `dumpcap` has stopped. The event-message is an unparsed error message from `dumcap` (a string).

SP_DROPS = 68

Reports the count of packets dropped in capture (an int).

SP_ERROR_MSG = 69

General error indicator; `dumpcap` has stopped. The event-message is an unparsed error message from `dumpcap` (a string).

SP_FILE = 70

`dumpcap` has recently opened a file to write newly captured packets. The event-message is the name of the file (a string).

SP_PACKET_COUNT = 80

Newly captured packets captured were written to the most recently given file. The event-message is the number of packets written (an int).

SP_SUCCESS = 83

General success indication, the event-message is None.

__weakref__

list of weak references to the object (if defined)

dropcount = None

The total number of packets received by `dumpcap`.

packetcount = None
The total number of packets dropped before `dumpcap` could receive them.

class `wirepy.lib.dumpcap.`**`Interface`**(*name*, *number=None*, *vendor_name=None*, *friendly_name=None*, *interface_type=None*, *addresses=None*, *loopback=None*)
Bases: `builtins.object`

An interface or device `dumpcap` can use to capture packets from.

`__str__`()
Equal to `name`

static `get_interface_capabilities`(*interface*, *monitor_mode=False*)
Query link-layer-types an interface supports.

Parameters

- **interface** – The name of the interface to query.
- **monitor_mode** – True if the interface shall be put into monitor-mode before querying available link-layer-types.

Returns A tuple with two members, the first indicating wether the interface supports monitor-mode, the second being a list of `LinkLayerType`.

classmethod `list_interfaces`()
Report the interfaces dumpcap knows about.

Raises `ChildError` if `dumpcap` returns an error.

Returns A list of `Interface`-instances.

IF_AIRPCAP = 1
The AirPcap-device

IF_DIALUP = 6
Dialup

IF_PIPE = 2
A pipe

IF_STDIN = 3
Standard input

IF_USB = 7
USB

IF_VIRTUAL = 8
Virtual

IF_WIRED = 0
Wired device (probably Ethernet/DOCSIS)

IF_WIRELESS = 5
Wireless

`__weakref__`
list of weak references to the object (if defined)

addresses = None
A list of strings representing the addresses the interface is bound to.

can_rfmon
True if this interface supports monitor-mode.

capabilities
The capabilities of this interface.

See `get_interface_capabilities()` for details.

interface_type = None
One of *IF_...* like `Interface.IF_WIRED`

loopback = None
True if the interface is a loopback

name = None
The name of the interface.

supported_link_layer_types
A list of supported link-layer-types.

class `wirepy.lib.dumpcap.`**`LinkLayerType`**(*dlt*, *name*, *description*)
Bases: `builtins.object`

Represents a link-layer-type as reported by `dumpcap`

`__str__`()
Equal to `name`

`__weakref__`
list of weak references to the object (if defined)

description = None
The human-friendly name

name = None
The short-name of this link-layer-type

class `wirepy.lib.dumpcap.`**`LiveInterfaceStats`**
Bases: `builtins.object`

Receive statistics on the number of packets received and dropped from all interfaces.

The iterator on instances of this class provides a convenient way to receive statistics as they arrive without busy-waiting

The context-manger ensures that the child-process is terminated when the context ends.

Both may be used in concert to produce a generator iterator that can be passed around and automatically terminates `dumpcap` once the instance is garbage-collected:

```
def stats():
    with LiveInterfaceStats() as s:
        for results in s:
            yield results

stats_iter = stats()
next(stats_iter)  # Launch dumpcap and get statistics
next(stats_iter)  # Get new statistics...
...
del stats_iter  # or gc/stats_iter.close(), dumpcap is terminated.
```

`__enter__`()

Returns The instance itself.

`__exit__`(*exc_type*, *exc_value*, *traceback*)
Kill `dumpcap` through a call to `terminate()`

`__getitem__`(*interface*)
Receive the current statistics for the given interface.

Parameters **interface** – The name of the interface

Returns A tuple of (*packets received*, *packets dropped*)

`__init__`()
Start capturing interface statistics.

Raises `ChildError` if `dumpcap` reported an error.

`__iter__`()
Wait for fresh statistics by calling `wait_for_tick()` and yield them. The tick-event is cleared **after** yielding to the caller; a new call to next() will probably block but return the newest results.

Returns A tuple of (*interface_name*, (*packets received*, *packets dropped*)).

`__len__`()

Returns The number of interfaces currently known.

`clear_tick`()
Clears the tick-event.

Calls to `wait_for_tick()` may block again after calling this.

`terminate`()
Kill dumpcap.

`wait_for_tick`(*timeout=None*)
Block until dumpcap reports fresh statistics.

Parameters **timeout** – If not None the call blocks up to that amount of seconds before raising NoEvents.

Raises `NoEvents` if no new data arrived after **timeout** has passed.

`__weakref__`
list of weak references to the object (if defined)

`interfaces`
A tuple of all currently known interface names.

`wirepy.lib.dumpcap.`**`DUMPCAP_BIN`** **= (‘dumpcap’,)**
Name (and default args) of `dumpcap` executable

`wirepy.lib.dumpcap.`**`DUMPCAP_CHECK_INTERVAL`** **= 1.0**
Timeout after which `dumpcap` is checked for being still alive while in a blocking call. Shorter timeouts consume more cpu-time but cause errors to be reported more quickly.

1.5 epan Module

class `wirepy.lib.epan.`**`Dissect`**(*cdata_obj=None*, *create_proto_tree=True*, *proto_tree_visible=True*)
Bases: `wirepy.lib.cdata.CDataObject`

Object encapsulation for type epan_dissect_t

static **`cleanup`**(*cdata_obj*)
releases resources attached to the packet dissection. DOES NOT free the actual pointer

`fake_protocols`(*fake_protocols*)
Indicate whether we should fake protocols or not

fill_in_columns (*fill_col_exprs=True*, *fill_fd_columns=True*)
fill the dissect run output into the packet list columns

static **free** (*cdata_obj*)
Free a single packet dissection.

This is basically the same as .cleanup() with another call to g_free() on the pointer.

static **init** (*cdata_obj*, *create_proto_tree*, *proto_tree_visible*)
initialize an existing single packet dissection

prime_dfilter (*dfp*)
Prime a proto_tree using the fields/protocols used in a dfilter.

run (*wtap*, *frame*, *column_info=None*)
run a single packet dissection

class wirepy.lib.epan.**ExtValueString** (*cdata*)
Bases: wirepy.lib.epan.FieldValue, wirepy.lib.cdata.CDataObject

A value_string_ext

class wirepy.lib.epan.**Field** (*init*)
Bases: wirepy.lib.cdata.CDataObject

A _header_field_info

abbrev
Abbreviated name of this field.

bitmask
Bitmask of interesting fields.

bitshift
Bits to shift.

blurb
Brief description of field.

display
One of BASE or field bit-width if FT_BOOLEAN and non-zero bitmask.

id_
Field ID.

name
Full name of this field.

parent
parent protocol

same_name_next
Next Field with same abbrev.

same_name_prev
Previous Field with same abbrev

strings
value_string, range_string or true_false_string, typically converted by VALS(), RVALS() or TFS(). If this is an FT_PROTOCOL then it points to the associated protocol_t structure

type_
Field type.

type_is_integer
True if type is one of FT_INT or FT_UINT

class `wirepy.lib.epan.`**`RangeValue`**(*value_min*, *value_max*, *string*)
Bases: `wirepy.lib.epan.FieldValue`

A range_string

`__ge__`(*other*)
x.__ge__(y) <==> x>=y

`__gt__`(*other*)
x.__gt__(y) <==> x>y

`__le__`(*other*)
x.__le__(y) <==> x<=y

class `wirepy.lib.epan.`**`StringValue`**(*cdata*)
Bases: `wirepy.lib.epan.FieldValue`

A value_string

`__ge__`(*other*)
x.__ge__(y) <==> x>=y

`__gt__`(*other*)
x.__gt__(y) <==> x>y

`__le__`(*other*)
x.__le__(y) <==> x<=y

class `wirepy.lib.epan.`**`TrueFalseString`**(*true_string*, *false_string*)
Bases: `wirepy.lib.epan.FieldValue`

A true_false_string

`wirepy.lib.epan.`**`cleanup_dissection`**()
extern void init_dissection

`wirepy.lib.epan.`**`init_dissection`**()
Initialize all data structures used for dissection.

1.6 `ftypes` Module

class `wirepy.lib.ftypes.`**`FieldType`**(*ftenum*)
Bases: `builtins.object`

A ftenum_t

ABSOLUTE_TIME
Absolute time

alias of `FT_ABSOLUTE_TIME`

BOOLEAN
Bool

alias of `FT_BOOLEAN`

BYTES
Raw bytes

alias of `FT_BYTES`

DOUBLE
Double

alias of `FT_DOUBLE`

ETHER
Ethernet

alias of `FT_ETHER`

ETHER_LEN
Ethernet

alias of `FT_ETHER_LEN`

EUI64
64-Bit extended unique identifier

alias of `FT_EUI64`

EUI64_LEN
eui64_len

alias of `FT_EUI64_LEN`

FLOAT
Float

alias of `FT_FLOAT`

FRAMENUM
Frame number

alias of `FT_FRAMENUM`

GUID
GUID

alias of `FT_GUID`

GUID_LEN
GUID

alias of `FT_GUID_LEN`

INT16
16 bit wide integer

alias of `FT_INT16`

INT24
24 bit wide integer

alias of `FT_INT24`

INT32
32 bit wide integer

alias of `FT_INT32`

INT64
64 bit wide integer

alias of `FT_INT64`

INT8
8 bit wide integer

alias of `FT_INT8`

IPXNET
IPX

alias of `FT_IPXNET`

IPXNET_LEN
IPX

alias of `FT_IPXNET_LEN`

IPv4
IPv4

alias of `FT_IPv4`

IPv4_LEN
IPv4

alias of `FT_IPv4_LEN`

IPv6
IPv6

alias of `FT_IPv6`

IPv6_LEN
IPv6

alias of `FT_IPv6_LEN`

NONE
Special

alias of `FT_NONE`

NUM_TYPES
The number of field types

alias of `FT_NUM_TYPES`

OID
OID

alias of `FT_OID`

PCRE
PCRE

alias of `FT_PCRE`

PROTOCOL
Protocol

alias of `FT_PROTOCOL`

RELATIVE_TIME
Relative time

alias of `FT_RELATIVE_TIME`

STRING
String

alias of FT_STRING

STRINGZ
String

alias of FT_STRINGZ

UINT16
Unsigned 16 bit wide integer

alias of FT_UINT16

UINT24
Unsigned 24 bit wide integer

alias of FT_UINT24

UINT32
Unsigned 32 bit wide integer

alias of FT_UINT32

UINT64
Unsigned 64 bit wide integer

alias of FT_UINT64

UINT8
Unsigned 8 bit wide integer

alias of FT_UINT8

UINT_BYTES
Raw bytes

alias of FT_UINT_BYTES

UINT_STRING
Raw bytes

alias of FT_UINT_STRING

value_from_unparsed(*s*, *allow_partial_value=False*)
Create a new Value from an unparsed string representation

__weakref__
list of weak references to the object (if defined)

name
The name of this FieldType

pretty_name
A more human-friendly name of this FieldType

class wirepy.lib.ftypes.**Type**(*cdata*)
Bases: wirepy.lib.cdata.CDataObject

A _ftype_t

class wirepy.lib.ftypes.**Value**(*cdata*)
Bases: builtins.object

A fvalue_t

__len__()
The length in bytes of this value. Falls back to the wire_size if the true length is not available

len_string_repr(*rtype*)
Returns the length of the string required to hold the string representation of the field value.

Returns -1 if the string cannot be represented in the given rtype.

The length DOES NOT include the terminating NUL.

new()
Allocate and initialize a Value

to_string_repr(*rtype=None*)
A human-readable string representation of this value. Raises OperationNotPossible if the value cannot be represented in the given rtype.

__weakref__
list of weak references to the object (if defined)

1.7 `glib2` Module

GLib2-related objects used by libwireshark.

class `wirepy.lib.glib2.`**SinglyLinkedListIterator**(*init*, *callable=None*, *gc=True*)
Bases: `wirepy.lib.cdata.CDataObject`

A singly-linked list (GSList).

__iter__()
Iterate of all data-items in the list.

next
The next item in the list.

class `wirepy.lib.glib2.`**String**(*string*)
Bases: `wirepy.lib.cdata.CDataObject`

A GString

static free(*cdata_obj*)
Frees the memory allocated for the GString.

allocated_len
Amount of allocated memory.

len
The length of the string.

`wirepy.lib.glib2.`**from_gchar**(*cdata*, *free=True*)
Build a python-string from a gchar*

1.8 `prefs` Module

`wirepy.lib.prefs.`**apply_all**()
Call the “apply”-callback function for each module if any of its preferences have changed.

`wirepy.lib.prefs.`**copy**(*src*)
Copy a set of preferences

`wirepy.lib.prefs.`**`read_prefs`**`()`

Read the preferences file, make it global and return a new Preferences-instance

`wirepy.lib.prefs.`**`write`**`(`*to_stdout=False*`)`

Write the global preferences to the user's preference-file; write to stdout if to_stdout is True.

1.9 `timestamp` Module

Functions to get/set the timestamp-type behaviour of wireshark.

exception `wirepy.lib.timestamp.`**`InvalidTimestampValue`**

Bases: `wirepy.lib.timestamp.TimestampError`

An invalid timestamp-type was used.

exception `wirepy.lib.timestamp.`**`TimestampError`**

Bases: `builtins.Exception`

Base-class for all timestamp-related errors.

`__weakref__`

list of weak references to the object (if defined)

`wirepy.lib.timestamp.`**`ABSOLUTE`**

Absolute

alias of `TS_ABSOLUTE`

`wirepy.lib.timestamp.`**`ABSOLUTE_WITH_DATE`**

Absolute with date

alias of `TS_ABSOLUTE_WITH_DATE`

`wirepy.lib.timestamp.`**`DELTA`**

Since previously captured packet

alias of `TS_DELTA`

`wirepy.lib.timestamp.`**`DELTA_DIS`**

Since previously displayed packet

alias of `TS_DELTA_DIS`

`wirepy.lib.timestamp.`**`EPOCH`**

Seconds (and fractions) since epoch

alias of `TS_EPOCH`

`wirepy.lib.timestamp.`**`NOT_SET`**

Special value, timestamp type not set

alias of `TS_NOT_SET`

`wirepy.lib.timestamp.`**`PREC_AUTO`**

Special value, automatic precision

alias of `TS_PREC_AUTO`

`wirepy.lib.timestamp.`**`PREC_FIXED_SEC`**

Fixed to-seconds precision

alias of `TS_PREC_FIXED_SEC`

`wirepy.lib.timestamp.`**`RELATIVE`**

Since start of capture

alias of `TS_RELATIVE`

`wirepy.lib.timestamp.`**`SECONDS_DEFAULT`**

.

alias of `TS_SECONDS_DEFAULT`

`wirepy.lib.timestamp.`**`SECONDS_HOUR_MIN_SEC`**

.

alias of `TS_SECONDS_HOUR_MIN_SEC`

`wirepy.lib.timestamp.`**`SECONDS_NOT_SET`**

.

alias of `TS_SECONDS_NOT_SET`

`wirepy.lib.timestamp.`**`UTC`**

UTC time

alias of `TS_UTC`

`wirepy.lib.timestamp.`**`UTC_WITH_DATE`**

UTC time with date

alias of `TS_UTC_WITH_DATE`

`wirepy.lib.timestamp.`**`get_precision`**`()`

Get the currently set timestamp-precision.

Returns an opaque integer, e.g. `PREC_FIXED_SEC`

`wirepy.lib.timestamp.`**`get_seconds_type`**`()`

Get the currently set seconds-type.

Returns an opaque int, e.g. of `SECONDS_DEFAULT`.

`wirepy.lib.timestamp.`**`get_type`**`()`

Get the currently set timestamp-type.

Returns an opaque integer, e.g. `NOT_SET`

`wirepy.lib.timestamp.`**`is_initialized`**`()`

Check if the globally used timestamp settings have been set.

Returns True if the timestamp-type and seconds-type are set.

`wirepy.lib.timestamp.`**`set_precision`**`(`*tsp*`)`

Set the globally used timestamp-precision.

Parameters **tsp** – A timestamp-precision constant like `PREC_FIXED_SEC`.

`wirepy.lib.timestamp.`**`set_seconds_type`**`(`*ts_seconds_type*`)`

Set the globally used timestamp-second-precision.

Params ts_seconds_type A timestamp-second-type, e.g. `SECONDS_DEFAULT`.

`wirepy.lib.timestamp.`**`set_type`**`(`*ts_type*`)`

Set the globally used timestamp-type.

Params ts_type A timestamp-type from this module, e.g. `RELATIVE`.

1.10 `wireshark` Module

Stub module

`wirepy.lib.wireshark.`**`mod`**

The cffi-module to libwireshark, libwsutils and libwtap

1.11 `wsutil` Module

`wirepy.lib.wsutil.`**`get_cur_groupname`**`()`

Get the current group or “UNKNOWN” on failure.

`wirepy.lib.wsutil.`**`get_cur_username`**`()`

Get the current username or “UNKNOWN” on failure.

`wirepy.lib.wsutil.`**`init_process_policies`**`()`

Called when the program starts, to enable security features and save whatever credential information we’ll need later.

`wirepy.lib.wsutil.`**`relinquish_special_privs_perm`**`()`

Permanently relinquish special privileges.

`init_process_policies()` must have been called before calling this.

`wirepy.lib.wsutil.`**`running_with_special_privs`**`()`

Return True if this program is running with special privileges.

`init_process_policies()` must have been called before calling this.

`wirepy.lib.wsutil.`**`started_with_special_privs`**`()`

Return True if this program started with special privileges.

`init_process_policies()` must have been called before calling this.

1.12 `wtap` Module

The wiretap-library is used to read capture files of various formats and encapsulation types.

exception `wirepy.lib.wtap.`**`BadFile`**(*err_info*, *for_writing*)

Bases: `wirepy.lib.wtap.FileError`

The file appears to be damaged or corrupted or otherwise bogus

exception `wirepy.lib.wtap.`**`CantClose`**(*err_info*, *for_writing*)

Bases: `wirepy.lib.wtap.FileError`

The file couldn’t be closed, reason unknown

exception `wirepy.lib.wtap.`**`CantOpen`**(*err_info*, *for_writing*)

Bases: `wirepy.lib.wtap.FileError`

The file couldn’t be opened, reason unknown

exception `wirepy.lib.wtap.`**`CantRead`**(*err_info*, *for_writing*)

Bases: `wirepy.lib.wtap.FileError`

An attempt to read failed, reason unknown

exception `wirepy.lib.wtap.`**`CantSeek`**`(`*err_info*, *for_writing*`)`
Bases: `wirepy.lib.wtap.FileError`

An attempt to seek failed, reason unknown

exception `wirepy.lib.wtap.`**`CantWriteToPipe`**`(`*err_info*, *for_writing*`)`
Bases: `wirepy.lib.wtap.FileError`

Wiretap can’t save to a pipe in the specified format

exception `wirepy.lib.wtap.`**`CompressionUnsupported`**`(`*err_info*, *for_writing*`)`
Bases: `wirepy.lib.wtap.FileError`

The filetype doesn’t support output compression

exception `wirepy.lib.wtap.`**`Decompress`**`(`*err_info*, *for_writing*`)`
Bases: `wirepy.lib.wtap.FileError`

Error decompressing

exception `wirepy.lib.wtap.`**`EncapPerPacketUnsupported`**`(`*err_info*, *for_writing*`)`
Bases: `wirepy.lib.wtap.FileError`

The specified format doesn’t support per-packet encapsulations

exception `wirepy.lib.wtap.`**`NotRegularFile`**`(`*err_info*, *for_writing*`)`
Bases: `wirepy.lib.wtap.FileError`

The file being opened for reading isn’t a plain file (or pipe)

exception `wirepy.lib.wtap.`**`RandomOpenPipe`**`(`*err_info*, *for_writing*`)`
Bases: `wirepy.lib.wtap.FileError`

The file is being opened for random access and it’s a pipe

exception `wirepy.lib.wtap.`**`RandomOpenStdin`**`(`*err_info*, *for_writing*`)`
Bases: `wirepy.lib.wtap.FileError`

We’re trying to open the standard input for random access

exception `wirepy.lib.wtap.`**`ShortRead`**`(`*err_info*, *for_writing*`)`
Bases: `wirepy.lib.wtap.FileError`

An attempt to read read less data than it should have

exception `wirepy.lib.wtap.`**`ShortWrite`**`(`*err_info*, *for_writing*`)`
Bases: `wirepy.lib.wtap.FileError`

An attempt to write wrote less data than it should have

exception `wirepy.lib.wtap.`**`UncompressBadOffset`**`(`*err_info*, *for_writing*`)`
Bases: `wirepy.lib.wtap.FileError`

LZ77 compressed data has bad offset to string

exception `wirepy.lib.wtap.`**`UncompressOverflow`**`(`*err_info*, *for_writing*`)`
Bases: `wirepy.lib.wtap.FileError`

Uncompressing Sniffer data would overflow buffer

exception `wirepy.lib.wtap.`**`UncompressTruncated`**`(`*err_info*, *for_writing*`)`
Bases: `wirepy.lib.wtap.FileError`

Sniffer compressed data was oddly truncated

exception `wirepy.lib.wtap.`**`UnknownFormat`**(*err_info*, *for_writing*)
Bases: `wirepy.lib.wtap.FileError`

The file being opened is not a capture file in a known format

exception `wirepy.lib.wtap.`**`Unsupported`**(*err_info*, *for_writing*)
Bases: `wirepy.lib.wtap.FileError`

Supported file type, but there's something in the file we can't support

exception `wirepy.lib.wtap.`**`UnsupportedEncap`**(*err_info*, *for_writing*)
Bases: `wirepy.lib.wtap.FileError`

Wiretap can't read or save files in the specified format with the specified encapsulation

exception `wirepy.lib.wtap.`**`UnsupportedFileType`**(*err_info*, *for_writing*)
Bases: `wirepy.lib.wtap.FileError`

Wiretap can't save files in the specified format

exception `wirepy.lib.wtap.`**`WTAPError`**
Bases: `builtins.Exception`

Base-class for all wtap-errors.

`__weakref__`
list of weak references to the object (if defined)

class `wirepy.lib.wtap.`**`EncapsulationType`**(*encap*)
Bases: `builtins.object`

An encapsulation type like "ether"

`__weakref__`
list of weak references to the object (if defined)

class `wirepy.lib.wtap.`**`PacketHeader`**(*cdata_obj*)
Bases: `wirepy.lib.cdata.CDataObject`

A wtap_pkthdr from wtap.h

`caplen`
Data length in the file.

`comment`
Optional comment.

`drop_count`
Number of packets lost.

`interface_id`
Identifier of the interface.

`len`
Data length on the wire.

`pkt_encap`
The EncapsulationType of the current packet.

`presence_flags`
What stuff do we have?

class `wirepy.lib.wtap.`**`WTAP`**(*cdata*)
Bases: `builtins.object`

A wtap from wtap.h

close()
Close the current file

fdclose()
Close the file descriptor for the current file

classmethod open_offline(*filename*, *random=False*)
Open a file and return a WTAP-instance. If random is True, the file is opened twice; the second open allows the application to do random- access I/O without moving the seek offset for sequential I/O, which is used by Wireshark to write packets as they arrive

sequential_close()
Close the current file

__weakref__
list of weak references to the object (if defined)

file_encap
The encapsulation-type of the file

file_size
The file-size as reported by the OS

file_type
The type of the file

is_compressed
True if the file is compressed (e.g. via gzip)

packetheader
The packet header from the current packet

read_so_far
The approximate amount of data read sequentially so far

tsprecision
The timestamp precision, a value like FILE_TSPREC_SEC

wirepy.lib.wtap.**iter_encapsulation_types**()
Iterates over all encapsulation-types wireshark can understand

wirepy.lib.wtap.**iter_file_types**()
Iterates over all file-types wireshark can understand

CHAPTER 2

Installation

2.1 Requirements

- CPython 3.x or later
- CFFI 0.6 or later
- Wireshark 1.10 or later
- GLib2 2.16.0 or later
- nose and tox are used for testing

2.2 Configuring Wireshark

- If you are using a Linux distribution, CPython-, Wireshark and their headers can be usually be installed from the package repository (e.g. via yum).
- Otherwise you may configure and build a **minimal** Wireshark library like this:

  ```
  ./configure -q --prefix=$HOME/wireshark --disable-wireshark --disable-packet-editor --disable-ed
  make -sj9
  make install
  ```

2.3 Configuring wirepy

1. Run `./configure` to configure *Wirepy*'s sourcecode:
   - Running `./configure` as it is should work if you have wireshark installed through *pkg-config*.
   - **Otherwise** you need to specify the paths to wireshark's and glib's header files yourself. You may also want to use a locally installed version of wireshark. The command may look something like this:

     ```
     DEPS_CFLAGS="-I/path/to/wireshark-headers -I/path/to/glib-2.0-headers" DEPS_LIBS="-L/path/to
     ```

     Executing may look like this:

     ```
     LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/path/to/wireshark/lib PATH=$PATH:/path/to/wireshark/bin ma
     ```

2. Take a look at the `Makefile` and use `make`.

CHAPTER 3

Development

Wirepy uses CFFI to create an interface to `libwireshark`, `libwsutil` and `libwiretap`. Class-based representations of the defined C-structs and -types are used to bind behavior, state and documentation. Instead of returning error values, all functions raise exceptions of appropriate type. Memory is handled by python's garbage collector most of the time.

The entire wireshark-interface can be found in `/lib`; one may need special knowledge about how to use classes there. Once things quiet down in `/lib`, a more pythonic API is to be created outside of `/lib`.

- What (at least in part) works:
  - Enumerating live interfaces and their capabilities.
  - Reading packets from live interfaces.
  - Reading packet dumps using the wiretap library.
  - Compiling and using display-filters to filter the resulting frame data.
  - Inspection of the resulting protocol-tree (`epan.ProtoTree`),
    - inspection of it's fields (`ftypes.FieldType`).
    - and their values (`epan.FieldValue`).
  - Working with columns, including `COL_CUSTOM`.
- What does not:
  - Putting it all together.
    - We probably want to create class-based representations of protocols, fields and their known values; one might create a class factory that uses the functions from `/lib` to create classes for protocols as they pop into existence in a proto-tree and keep a weakref to those.
    - It should be fairly easy to use the above for class-based comparision of values and create a simple compiler for display-filter strings (e.g. "`DisplayFilter(IP.proto==IP.proto.IPv4)`").
    - A `FieldType` should have it's own subclass that is able to interpret common python objects, preserving it's type as closely as possible.
      - A `INT8` should do arithmetic mod 2**8
      - A `IPv4` or `IPv6` may take values from the `ipaddr`-module
      - etc

    This should live outside of `/lib`.
  - Writing packet dumps through `wtap_dump...`.

  - Taps and the other ~95% of the more useful functions of wireshark.
  - Plugins will not load because they expect the symbols from `libwireshark` in the global namespace. We hack this situation by flooding the namespace with a call to dlopen().
  - A backport to Python 2.x (using a compat module) should be easy.

- To be considered:
  - There are many ways in which `libwireshark` handles memory allocation. From within Python, everything should be garbage-collected though;
  - There are many ways in which `libwireshark` handles memory deallocation. Once some or the other function is called or state is reached, memory represented by reachable objects becomes invalid garbage.
  - The raw C-API very much expects C-like behavior from it's user; there are many de-facto global states and carry-on-your-back variables. Hide those

CHAPTER 4

Contact

Via lukas.lueg@gmail.com. Please use github to report problems and submit comments (which are very welcome). Patches should conform to **PEP 8** and have appropriate unittests attached.

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

Generated January 26, 2014.

Python Module Index